Searchlight: Won't You Be My Neighbor?

Mehedi Bakht
Dept. of Computer Science
University of Illinois
Urbana-Champaign
mbakht2@illinois.edu

Matt Trower
Dept. of Computer Science
University of Illinois
Urbana-Champaign
mtrower2@illinois.edu

Robin Kravets
Dept. of Computer Science
University of Illinois
Urbana-Champaign
rhk@illinois.edu

ABSTRACT

The rapid deployment of millions of mobile sensors and smartphones has resulted in a demand for opportunistic encounter-based networking to support mobile social networking applications and proximity-based gaming. However, the success of these emerging networks is limited by the lack of effective and energy efficient neighbor discovery protocols. While probabilistic approaches perform well for the average case, they exhibit long tails resulting in high upper bounds on neighbor discovery time. Recent deterministic protocols, which allow nodes to wake up at specific timeslots according to a particular pattern, improve on the worst case bound, but do so by sacrificing average case performance. In response to these limitations, we have designed Searchlight, a highly effective asynchronous discovery protocol that is built on three basic ideas. First, it leverages the constant offset between periodic awake slots to design a simple probing-based approach to ensure discovery. Second, it allows awake slots to cover larger sections of time, which ultimately reduces total awake time drastically. Finally, Searchlight has the option to employ probabilistic techniques with its deterministic approach that can considerably improve its performance in the average case when all nodes have the same duty cycle. We validate Searchlight through analysis and real-world experiments on smartphones that show considerable improvement (up to 50%) in worst-case discovery latency over existing approaches in almost all cases, irrespective of duty cycle symmetry.

Categories and Subject Descriptors

C.2.1 [**Computer Systems Organization**]: COMPUTER-COMMUNICATION NETWORKS— *Network Architecture and Design*

Keywords

Neighbor Discovery, Ad-hoc Networks, Smartphone, Mobile Social Networks

MobiCom'12, August 22–26, 2012, Istanbul, Turkey.

1. INTRODUCTION

The world of social networking is being transformed by mobile devices. Where early applications required users to check into centralized servers to find and coordinate with friends (e.g., Google Latitude/DodgeBall [1], Gowalla [2], Foursquare [3], FaceBook Places [4], etc.), new applications are being built around the local communication capacity of all smartphones (e.g., Who's Near Me [5] MobiClique [6], , Aka-aki [7], BlueHoo [8]). One of the fastest growing areas of interest is the mobile gaming community, where users look for and play games with other players in their wireless network vicinity (e.g., Nintendo's StreetPass [9] and Sony's Vita [10]). Although the popularity of these proximity-based applications has increased, the network services needed to enable effective and energy-efficient performance have not caught up - especially when users want to operate in a purely local fashion. Essentially, the success of such applications depends on the availability of services that provide information about nearby users, or, more specifically, nearby users' devices.

While centralized services can provide much of the support needed for proximity-based applications, centralized services have significant problems. First, if the service is unavailable, the users are out of luck, even if they can see other users in their vicinity. Second, either a centralized location service must push location information to all users, or users must pull location information as they need it, resulting in excessive updates or control overhead. Third, most applications run their own proprietary location service, requiring users to register with many applications. Fourth, many users are reluctant to expose their location information to a centralized service, but may be willing to let people that are in their general vicinity or that can already see them discover their device and actual location. Finally, communicating through a central server instead of directly to near-by neighbors can introduce significant communication latency, limiting its usefulness for real-time and gaming applications. Therefore, local solutions are needed that do not rely connectivity to centralized servers and services.

To enable the full potential of these proximity-based applications, the applications need local services that find the direct neighbors in a device's communication range. Although neighbor discovery is also challenging in environments such as ad hoc and sensor networks, there is some expectation of network stability, so applications expect connectivity and deal with failure as it happens. There is also some expectation of sufficient density so that when a device has data to send, some number of neighbors will be available at the

time of discovery. However, social networks do not have the same expectation of stability or device density. Similar to devices in a delay-tolerant network, devices in a social network search for neighbors and then decide to communicate or send data. Since most smartphones come with both Bluetooth and Wi-Fi and given the extreme differential in the energy needed for discovery between Bluetooth and Wi-Fi, most (if not all) current proximity-based applications limit proximity detection to the cheaper Bluetooth radio. (e.g., Aka-aki and Bluehoo). However, limiting neighbors to be within Bluetooth range obviously limits the reach and effectiveness of social networking applications. While there are many energy-efficient MAC protocols, there is still the need for more energy-efficient neighbor discovery protocols for higher-power communication technologies like Wi-Fi.

The key goal for energy-efficient neighbor discovery is to reduce energy consumption, while still enabling effective neighbor discovery, where effectiveness can be determined by both the ability to discover a neighbor as well as the latency to discovery that neighbor. While the impact of fewer neighbor discoveries is intuitive, discovery latency has some less obvious impacts. In stationary networks, high discovery latency can result in longer set up times, but does not adversely affect the normal operation of the network. In mobile environments, high discovery latencies can result in missing short, yet meaningful contact opportunities, resulting in reduced effectiveness of the application. Unfortunately, current discovery protocols, mostly aimed at sensor networks, have not been able to ensure both low average discovery latency while still guaranteeing a low maximum bound on that latency.

In response to these limitations, we have designed Searchlight, a highly effective and energy-efficient duty-cycle-based neighbor discovery protocol that provides improvements in average and worst-case discovery latency while still providing a significant decrease in energy consumption. Searchlight is designed around the observation that two energy conserving devices follow well-defined patterns to enable discovery, resulting in a stable relationship between the patterns of any two devices, which allows Searchlight to significantly reduce the worst-case bound on discovery latency. By further employing a random/probabilistic component to the protocols, Searchlight considerably reduces average discovery latency as well. By slightly increasing each awake time, Searchlight can eliminate a significant number of redundant discoveries to reduce total awake time, and so total energy consumption for discovery, by almost 50%. Finally, Searchlight can be applied to devices with different duty cycles if all of the chosen duty cycles are a power of two multiple of the smallest duty cycle (i.e., x, $2x$, $4x$, $8x$, ...).

To validate the design and feasibility of Searchlight, we evaluated it through simulation and tested a real implementation on commercial off-the-shelf hardware. Our results show that Searchlight significantly reduces the worst-case discovery latency bound by 50% for a given energy budget. Searchlight also improves average discovery latency by up to 25% for low duty cycles. To gauge the potential of Searchlight in a real network setting, we implemented Searchlight on a testbed consisting of Nokia N900 smartphones. Since most recent duty-cycle-based neighbor discovery algorithms were targeted at and implemented on sensor motes (i.e., U-Connect [11]), our implementation provides valuable new insight into issues related to the execution of such protocols on smartphones with Wi-Fi interfaces to support mobile social networking applications. However, it is important to note that Searchlight is a general discovery protocol that provides significantly improved performance in both mobile and sensor environments.

The rest of this paper is organized as follows. Section 2 briefly describes existing approaches to neighbor discovery and discusses why they fail to meet the performance goals. Section 3 describes Searchlight in detail, including analysis that shows how it improves the worst case bounds for the symmetric case and how this improves the average latency for both cases. Section 4 presents our simulation-based performance evaluation of Searchlight in comparison to other neighbor discovery protocols. A prototype implementation of Searchlight on a smartphone testbed is described in Section 5. We conclude and outline the direction of our future research in Section 6.

2. ENERGY-EFFICIENT NEIGHBOR DISCOVERY FOR PROXIMITY-BASED APPLICATIONS

The key to all proximity-based applications is continuously determining who is near each other, even if the phone is not being used and is in a user's pocket. Although neighbor discovery services are being integrated into smartphone platforms - as seen by the emergence of Qualcomm's AllJoyn proximity-enhanced mobile software environment [12] and the integration of proximity-based services into the latest version of the Android OS [13] using Wi-Fi-Direct [14] - the challenge still remains of balancing battery life and connectivity. In general, the more devices these applications can discover, the better they can perform. However, for a device running on a battery-and thus on a limited energy budget, it is not practical to continuously search for neighbors. Existing approaches such as Bluetooth's inquiry method were designed with user initiated requests in mind and do not fare well under continuous discovery. A more feasible approach is to adopt the techniques from the sensor network community and keep the wireless interface in a sleep state most of the time and periodically wake it up for discovery.

The success of such duty-cycling schemes depends on ensuring that the wakeup times of two neighboring devices overlap. This is not hard to achieve when device clocks can be synchronized, for example through GPS [15, 16]. However, synchronization through GPS is usually too energy-expensive for mobile sensors [17] and smartphones [18]. This has necessitated the design of periodic schemes that ensure such overlap within a reasonable time bound while operating at low duty-cycles and requiring *no clock synchronization*.

While there are many asynchronous communication protocols (e.g., BMAC [19], SMAC [20]), the majority of these assume that all devices (which we call nodes from here on) have *symmetric* sleep patterns (i.e., run at the same duty cycle). However, realistic scenarios, for both mobile social networking and sensor networks, are more likely to result in nodes with varying energy requirements and so *asymmetric* duty cycles. While nodes may start with the same duty cycle, as the network is used, the available energy to each node will change, resulting in asymmetric duty cycles. In response, a new class of asynchronous and asymmetric slotted discovery protocols has evolved.

Asynchronous neighbor discovery algorithms mostly work on a time-slot basis, where time is assumed to be divided

into slots of equal size and all nodes agree on the size of a slot. Based on the protocol used, nodes decide to remain awake during specific slots, called *active* slots, and sleep during the remaining slots. During an active slot, the node may send/receive or do both, depending on application requirements. Successful discovery takes place between two neighboring nodes whenever their active slots overlap. To be energy efficient, a discovery scheme needs to use as few active slots as possible to discover neighbors within a reasonable time limit. Current approaches to energy-efficient asynchronous neighbor discovery fall broadly into two categories - probabilistic and deterministic. The relative strengths and weaknesses of these existing schemes can be judged by comparing their average and worst-case latency, as well as considering how well they handle duty-cycle asymmetry.

Both average and worst-case latency are important to opportunistic applications. While it is clear that these applications can benefit from a reduced average latency, most such applications are designed to handle missed contacts and so may seem like they would not benefit from a lower worst-case bound. However, successful discoveries still determine the effectiveness of these applications. Since many encounters may be short, lower worst-case bounds can enable much higher discovery success ratios, and so more contact opportunities.

Most well-known among probabilistic approaches is a family of "birthday protocols" [21] where nodes transmit/receive or sleep with different probabilities. Given the probabilistic nature of this type of approach, such scheme work well in the average case and allow asymmetric operation, since there is no restriction on duty cycle length. However, the main drawback of the birthday protocol is its failure to provide a bound on the worst case discovery latency, leading to long tails on discovery probabilities.

Deterministic protocols, by either ensuring a "quorum" or using prime values for duty cycle lengths, provide strict guarantees on worst-case discovery latency. In the Quorum-based protocols [22, 23], time is divided into sets of m^2 contiguous intervals. These m^2 intervals are arranged as a 2-dimensional $m \times m$ array and each host can pick one row and one column of entries as awake intervals. This pattern ensures that no matter what row and column are chosen, any two nodes have at least two overlapping awake intervals. While the Quorum protocol provides a reasonable bound on worst-case latency, it performs much worse than the probabilistic approach in the average case. Since m is a global parameter, the initial Quorum protocol [23] only supports symmetric operation. Lai et al. [22] provided an improvement to handle asymmetric cases when there are only two different schedules in the entire network. However, allowing only two duty cycles, and so only two energy-saving levels, is too restrictive. Another approach that only works for the symmetric case is the application of block design using difference sets [24]. For the asymmetric case, designing the appropriate schedule following the proposed scheme becomes similar to the vertex-cover problem, which is an NP-complete problem. Hence, these kinds of approaches are limited to symmetric networks.

Prime-based deterministic protocols overcome this limitation and can handle both symmetric and asymmetric operation while still providing a strict bound on worst-case latency. In Disco [25], each node chooses a pair of prime numbers such that the sum of their reciprocals is as close as possible to the desired duty cycle. The nodes then wake up at multiples of the individual prime numbers. If one node chooses primes p_1, p_2 and another node chooses p_3, p_4, the worst-case discovery latency between these two nodes will be $min\{(p_1 \cdot p_3), (p_1 \cdot p_4), (p_2 \cdot p_3), (p_2 \cdot p_4)\}$, provided the two primes in the pair are not equal. A more recent deterministic approach, U-Connect [11], uses a single prime per node. Instead of just waking up only 1 slot every p slots, the nodes also wake up $\frac{p+1}{2}$ slots every p^2 slots. The worst-case latency for U-Connect is p^2 which is similar to Disco. However, for the energy-latency product, a metric proposed by the authors to evaluate the energy-efficiency of asynchronous neighbor discovery protocols, U-Connect fares better than Disco in the symmetric case.

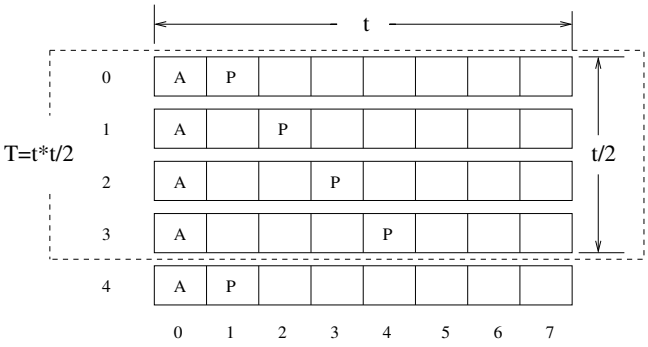

Figure 1: Searchlight with sequential probing (t=8)

Although these deterministic protocols have good worst-case performance, in the average case, they are considerably worse than the birthday protocol. To successfully meet all of the goals of energy-efficiency, asymmetry, and good average-case and worst-case latency, we present a new protocol named Searchlight. Searchlight follows a deterministic approach and provides a strict bound on worst-case latency, which is *lower* than existing protocols for both symmetric and asymmetric cases. While deterministic protocols guarantee discovery within a certain bound by using a particular pattern to determine the awake slots, these protocols overlook hidden sub-patterns within these patterns. When the awake slots occur periodically, Searchlight leverages the constant offset between the awake slots of any two nodes using the same period. Additionally, the best performing protocols frequently result in multiple discoveries, where one is sufficient. Searchlight reduces these extraneous discoveries, resulting in a 50% reduction in active slots in symmetric scenarios and only slightly less in asymmetric scenarios. Finally, Searchlight incorporates randomization techniques that significantly improve average discovery latency in the all symmetric cases or even when only some nodes as symmetric.

3. SEARCHLIGHT

Searchlight is an asynchronous periodic slot-based discovery protocol, where a period consists of t contiguous slots as determined by the target duty cycle for a given node. Nodes sleep during the majority of the slots and can only discover each other if they are active during the same slot. Searchlight can operate in two modes: symmetric, where all nodes have the same t, and asymmetric, where nodes may have different ts. The insight into the design of Search-

light comes from the very simple observation that when two nodes wake up periodically with the same interval, the temporal distance between these awake slots always remains the same (assuming negligible clock drift, which we discuss at the end of this section). Searchlight leverages this constant temporal relationship to provide better worst-case bounds on discovery latency than any existing protocol, and then improves the energy cost further by reducing the number of active slots required to ensure discovery by almost half by allowing active slots to "overflow". In addition, Searchlight also has the option to employ probabilistic techniques that considerably improves discovery latency in the average case when all nodes operate on same duty cycle.

In this section, we describe Searchlight in detail, explaining the design of systematic probing, "overflowing" awake slots, and the introduction of randomized probing. Through analysis, we derive the worst-case latency bounds for both the symmetric and asymmetric cases and show that Searchlight reduces worst-case discovery latency by 50% in the symmetric case.

3.1 Systematic Probing

In Searchlight, each node has two active slots in t. The first active slot, called the *anchor* slot, is the first slot in the period. In the symmetric case, since the position of this anchor slot is fixed in t but the start times for the ts for different nodes vary, the anchor slots for two nodes only overlap if the difference between the start times of the two periods is less than one timeslot. For all other offsets, assuming negligible clock drift, the two anchor slots would never meet since the offset remains constant. Essentially, the relative position of the anchor slot of one node always remains the same with respect to that of the other node and is in the range $[1, t-1]$. Since most anchor slots miss each other, Searchlight introduces a second active slot in t called the *probe* slot, which systematically searches for the anchor slot of the other node. It is important to note that anchor slots and probe slots perform exactly the same operation - the different names are used to merely convey the fact that one remains fixed in each period, while the other one moves around in search of the anchor slot of the other node. Hence, probe-anchor, probe-probe and anchor-anchor overlaps all result in discovery.

Given the stability of the relative offset, no two nodes can be offset by more than $\frac{t}{2}$ slots. Therefore, Searchlight only needs to probe the first half of the t slots to guarantee an overlap between the probe slot of one node and the anchor slot of the other node. To achieve this goal, the position of the probe slot is determined by a counter that starts at 1, increments by 1 every period, ends at $\lfloor \frac{t}{2} \rfloor$ and then starts at 1 again (see Figure 1). In other words, if P_i denotes the position of the probe slot in the i-th period, then

$$P_{i+1} = ((P_i) \mod \lfloor \frac{t}{2} \rfloor) + 1. \quad (1)$$

The position of the probe slot actually follows the periodic pattern $\{1, 2, ..., \lfloor \frac{t}{2} \rfloor\}$ and this pattern gets repeated every $\lfloor \frac{t}{2} \rfloor$ periods, which we call the *hyper-period* T. For example, for $t = 8$, Searchlight uses the periodic pattern {1,2,3,4} to determine the position of the probe slot in each period.

3.2 Discovery Latency

In this section, we present the analysis for worst-case discovery latency for Searchlight with symmetric duty cycles.

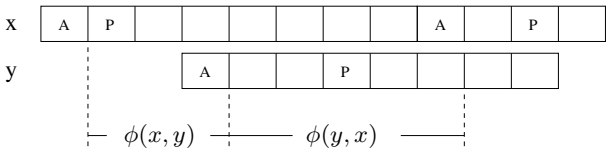

Figure 2: Phase offsets (t=8)

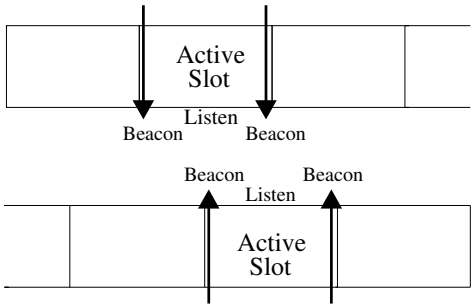

Figure 3: Beaconing in an active slot

LEMMA 3.1. *The worst-case discovery latency for Searchlight with parameter t is equal to $t \cdot \lfloor \frac{t}{2} \rfloor$ slots.*

Proof: For two nodes x and y, let $\phi(x, y)$ be the phase offset (in slots) from the anchor slot of x, which we call A_x, to the anchor slot of y, which we call A_y. Similarly, let $\phi(y, x)$ be the phase offset from A_y to A_x (see Figure 2). Clearly,

$$\phi(x, y) + \phi(y, x) = t. \quad (2)$$

In the symmetric case, where both nodes use the same t, $\phi(x, y)$ and $\phi(y, x)$ remain constant during the contact. It follows from Equation (2) that

$$min(\phi(x, y), \phi(y, x)) \leq \lfloor \frac{t}{2} \rfloor.$$

The probe slot goes through all positions from 1 to $\lfloor \frac{t}{2} \rfloor$ every $\lfloor \frac{t}{2} \rfloor$ periods. Let us denote the probe slots of x and y as P_x and P_y respectively. P_x will meet A_y within $\lfloor \frac{t}{2} \rfloor$ periods as long as

$$1 \leq \phi(x, y) \leq \lfloor \frac{t}{2} \rfloor.$$

Similarly, P_y will meet A_x within $\lfloor \frac{t}{2} \rfloor$ periods as long as

$$1 \leq \phi(y, x) \leq \lfloor \frac{t}{2} \rfloor.$$

So, as long as

$$min(\phi_(x, y), \phi_(y, x)) \leq \lfloor \frac{t}{2} \rfloor,$$

discovery will take place within $\lfloor \frac{t}{2} \rfloor$ periods or $t \cdot \lfloor \frac{t}{2} \rfloor$ slots.

3.3 Slot non-alignment

For illustration purposes, we have shown the slots to be aligned. However, Searchlight is designed as an asynchronous protocol and hence does not assume or rely on the alignment of slot boundaries at different nodes. To ensure that an overlap between two active slots always leads to discovery, Searchlight employs the same beaconing strategy as

Disco [25]. A beacon gets sent both at the beginning and end of an active slot and the node remains in listening mode in the intermediate period (see Figure 3). Because of this approach, non-alignment of slot boundaries actually results in lower discovery latency in general, since each slot overlaps with two slots of the other node and discovery can happen from either of the two overlaps.

However, two overlaps are redundant when discovery can be ensured with just one overlap. Based on this observation, we next describe striped probing, a strategy that improves the performance of Searchlight significantly by eliminating redundant probes.

3.4 Striped probing

Probing of every slot from 1 to $\lfloor \frac{t}{2} \rfloor$ guarantees two overlaps every $\lfloor \frac{t}{2} \rfloor$ periods whenever the slot boundaries of two nodes are not aligned. To reduce this redundancy, the probe slot can instead probe every even slot, by using a counter that starts at 2, increments by 2 every period up to $(2 \cdot \lceil \frac{\lfloor \frac{t}{2} \rfloor}{2} \rceil))$ and then starts at 2 again. In other words, if P_i denotes the position of the probe slot in the i-th period, then

$$P_{i+1} = ((P_i) \mod (2 \cdot \lceil \frac{\lfloor \frac{t}{2} \rfloor}{2} \rceil)) + 2. \quad (3)$$

It is important to note that striping cannot be applied to the birthday protocol or Disco since neither of these protocols use consecutive slots. Since U-Connect has an additional set of $\frac{p+1}{2}$ sequential active slots every p^2 slots, striping can be used to reduce this to $\frac{p+1}{4}$, resulting in 10% fewer active slots. We evaluate this striped version of U-Connect in our implementation.

However, striped probing would not work for either protocol in the rare case when slot boundaries are completely aligned. To handle perfect alignment, each active slot "overflows" by δ, a small amount that is sufficient to receive a leading beacon for another node. This basically means that if a regular slot size is x, then every active slot (both anchor and probe) become $x(1+\delta)$ long, and length of the regular slot that follows is reduced to $x(1-\delta)$. The value of δ should be the smallest possible overlap that guarantees discovery. Because of factors like message transmission time, clock drift, etc., the specific value of δ will be largely platform-dependent. As a guideline, δ should be a small fraction of the slot size (i.e., 1%) or a fixed minimum value based on the device specifications.

The main advantage of striped probing is that it significantly reduces the worst-case bound. Using the approach detailed in Section 3.2, it is easy to show that the worst-case bound for the symmetric case changes to $t \cdot \lceil \frac{\lfloor \frac{t}{2} \rfloor}{2} \rceil$ slots.

3.5 Asymmetry

The main design of Searchlight is based on the constant relative offset between the anchor slots of any two neighboring nodes. Unfortunately, this constant offset does not hold in the asymmetric case, when nodes can choose different values of t based on their own energy requirements.

One possible approach to guarantee overlap withing a fixed bound is to require that t should always be prime. Essentially, restricting t to be a prime number, similar to Disco and U-Connect, would ensure that for any two nodes operating at different duty cycles, their period lengths t_1 and t_2 will be relatively prime, i.e., they will have no common factors other than 1. Since period lengths are relatively prime, it follows from the Chinese Remainder Theorem [26] that the two anchor slots would overlap at least once every $t_1 \cdot t_2$ slots, where t_1 and t_2 are the two different period lengths. However, restricting t to prime number does not maintain the necessary constant offset.

Instead of using primes, Searchlight takes a novel approach to duty cycle asymmetry. It is possible to maintain the constant offset for the anchor slots of any two nodes if the larger valued period is an integer multiple of the smaller one. To see how this works, consider two nodes, node A with period $3t$ and node B with period t (see Figure 4). B's anchor slots always have the same relative distance from all of A's anchor slots. Since A only probes half of its slots, this ensures that the probe slot of A will eventually meet with an any anchor slot of B that falls in the first half of A's cycle. In this example, B always probes the first $3t/2$ slots. Since A's anchor slots always align with slots 2 and 10 for B, discovery is guaranteed. This example can be generalized to A having any value nt.

While any multiple works for a given pair of nodes, to maintain the constant offset for all nodes, Searchlight restricts the duty cycle to a power-multiple of the smallest duty cycle (i.e. 2, 4, 8, 16 or 3, 9, 27, 81, etc.), guaranteeing that any two nodes' duty cycles are multiples of each other. This means that if t is the smallest duty cycle, the other nodes can select $2t$, $4t$, $8t$, etc, for any t. Although this seems to restrict the choice of energy conservation levels, it is important to note that the use of primes is also very restrictive. While prime-based approaches are restricted to only primes, Searchlight can be set up to use any base value of t to fit the requirements of the system. Additionally, the use of the power of 2 multiples provides the most flexibility at the shorter duty cycles, giving applications more choices where it is needed. Finally, Searchlight can use any power of x multiple, providing a little more flexibility if necessary.

Along with flexibility, the power-multiple-based approach also provides the best worst-case latency bound, which is based on the time needed for A's probe slot to meet an anchor slot of B. A's probe slot probes every other slot from 2 to $\lceil \frac{\lfloor \frac{t_A}{2} \rfloor}{2} \rceil$. Let us assume that within that range, there are m anchor slots of B. In the worst-case, when A meets B, A's probe slot would be just past the $m-th$ anchor slot. To overlap, the probe slot will still have to go up to $\lceil \frac{\lfloor \frac{t_A}{2} \rfloor}{2} \rceil$, and then start from 2 again and reach the 1st anchor slot of B. That means the probe slot would have to travel $t_B + (\lceil \frac{\lfloor \frac{t_A}{2} \rfloor}{2} \rceil \mod t_B)$ slots. With striped probing, it would require

$$\frac{t_B + (\lceil \frac{\lfloor \frac{t_A}{2} \rfloor}{2} \rceil \mod t_B)}{2}$$

periods, or

$$\frac{(t_B + (\lceil \frac{\lfloor \frac{t_A}{2} \rfloor}{2} \rceil \mod t_B))}{2} \cdot t_A$$

slots.

It is possible to combine the prime-based and power-multiple-based approaches if necessary, as long as the chosen primes are relatively prime with the base t. For example, when four different duty cycles are needed, three allowable t val-

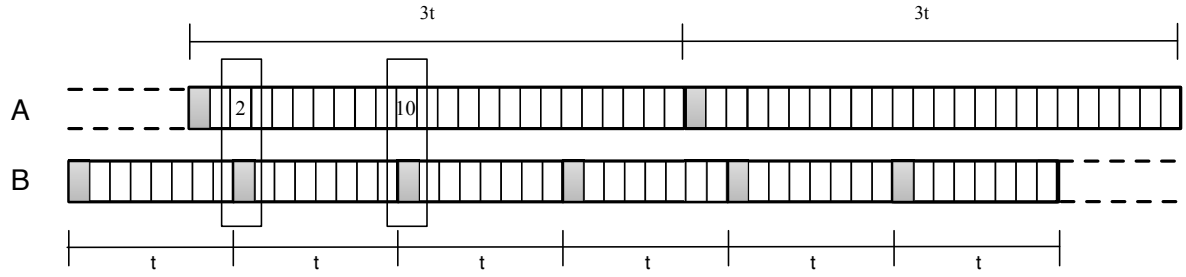

Figure 4: Searchlight with Asymmetric Duty Cycles (t and $3t$)

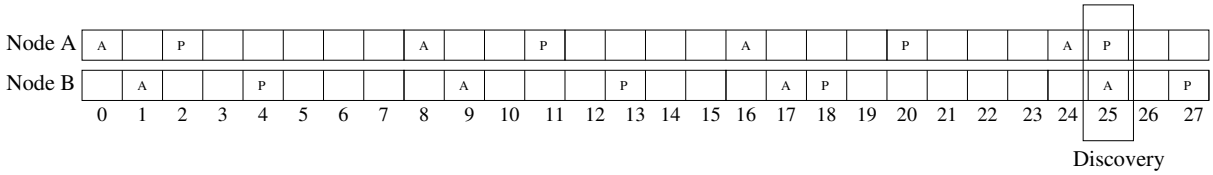

Figure 5: Overlap with sequential probing (t=8)

Protocol	Parameters	Duty Cycle	Worst-case bound Symmetric case
Disco	p_1, p_2	$\frac{p_1+p_2}{p_1 \cdot p_2}$	$p_1 \cdot p_2$
U-Connect	p	$\frac{3p+1}{2p^2}$	p^2
Searchlight	t	$\frac{2\cdot(1+\delta)}{t}$	$t \cdot \lceil \frac{\lfloor \frac{t}{2} \rfloor}{2} \rceil$

Table 1: Deterministic Protocols

Protocol	Worst-case bound(slots)
Disco	$4x^2$
U-Connect	$\frac{9x^2}{4}$
Searchlight	$2x(1+\delta))\lceil \frac{\lfloor \frac{2x(1+\delta)}{2} \rfloor}{2} \rceil \approx x^2(1+\delta)^2$

Table 2: Worst-case bounds for different protocols operating at the same duty cycle

ues can be $t_{base}, 2 \cdot t_{base}, 4 \cdot t_{base}$ and the fourth value can be any *prime* other than 2 that is relatively prime with t_{base}.

3.6 Duty Cycle and Energy Consumption

So far we have looked at the worst-case bounds on the discovery latency provided by Searchlight for different scenarios. However, it is important to consider the energy cost of achieving a particular bound. Since the primary cost we are concerned about is energy, we can look at the duty cycle. For a given t, there are two active slots every t slots. Taking into consideration the cost of extending the slots by δ, the duty cycle for any given t is $\frac{2\cdot(1+\delta)}{t}$. Table 1 summarizes the cost (duty cycle) and performance (worst-case bound) of the three main deterministic protocols. However, based on that information, it is hard to compare the different protocols since they use different parameters.

To provide a common baseline, it is interesting to keep the duty cycle fixed so that all protocols consume energy at the same rate and then look at the worst-case bound given by the different protocols. By doing so, we can compare the performance of different protocols when they incur the same cost. Disco does the best in the symmetric case when the two primes p_1 and p_2 are as close as possible. For ease of analysis, let us assume that $p_1 = p_2 = p$. Next, we express the parameters of each protocol as functions of x, where $\frac{1}{x}$ is the common duty cycle.

For Disco,

$$\frac{p_1 + p_2}{p_1 \cdot p_2} \approx= \frac{p+p}{p \cdot p} = \frac{1}{x}$$

and hence, $p = 2x$. Similarly, for Searchlight,

$$t = 2x(1+\delta)$$

. Finally, for U-Connect,

$$\frac{3p+1}{2p^2} = \frac{1}{x}$$

and so $p \approx \frac{3x}{2}$. Now, we can express the worst-case bounds for all protocols as $f(x)$ (see Table 2). It is clear from the table that even with a conservative value of $\delta = 0.1$ ($\frac{1}{10}$-th of the actual slot size), Searchlight reduces the worst case bound by around 50% in comparison to U-Connect, the current best performing deterministic protocol.

3.7 Randomized Probing

Sequential probing does a good job of ensuring discovery by finding the anchor slot of the other node (i.e., probe-anchor overlap). However, the overlap of two probe slots should also result in a successful discovery. While probe-probe overlap is possible in sequential probing, the probe slots of two nodes follow the same pattern, and hence they are often in sync with each other, greatly reducing the probability of a probe-probe overlap (see Figure 5). Consider two nodes A and B with a relative phase offset of one slot

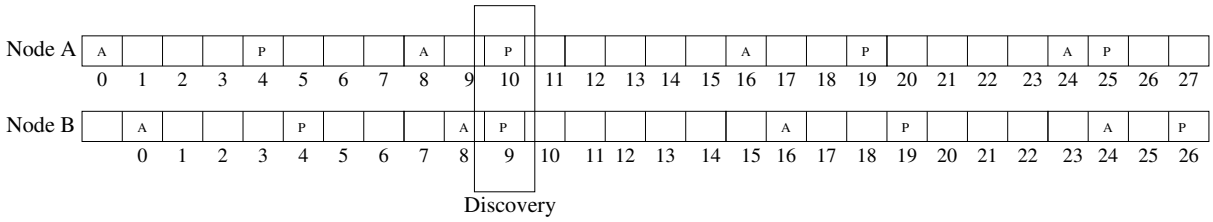

Figure 6: Overlap with randomized probing (t=8)

and $t = 8$. For sake of simplicity, let us assume that they are not using striped probing. When they first meet, A's probe slot is at position 2, while B's is at position 3 (with respect to its own anchor slot). Since both A and B follow the pattern {1,2,3,4}, the next positions of the probe slots will be 3 and 4 respectively. Thus, A and B's probe slots "chase" each other without ever overlapping at any point in time.

To increase the probability of a probe-probe overlap, Searchlight introduces a probabilistic component, similar to the Birthday protocols. Instead of being restricted to the pattern $1, 2, 3...\lfloor \frac{t}{2} \rfloor$, nodes can *randomly* pick any probe slot pattern that is a permutation of values from 1 to $\lfloor \frac{t}{2} \rfloor$. With striped probing, the range will change to $2, 4...(2 \cdot \lceil \frac{\lfloor \frac{t}{2} \rfloor}{2} \rceil))$. To differentiate, we call this version of Searchlight with randomized probing Searchlight-R, while the one with sequential probing is called Searchlight-S in the rest of the paper.

In the context of the last example, instead of being restricted to just {1,2,3,4}, Searchlight-R allows nodes to randomly choose any pattern that is a permutation of the integers 1,2,3 and 4 (e.g., {1,4,3,2}, {1,2,4,3}). In this case, A randomly chooses the periodic pattern {1,4,2,3} while its neighbor B chooses the periodic pattern {1,3,2,4} (see Figure 6). When they first meet, the probe slots of A and B are at positions 4 and 4 respectively and their relative offset is 1 slot. Because of the phase offset, the probe slots miss each other initially but meet in the next period when A's probe slot moves to position 2 and B's probe slot moves to position 1. Thus, the use of different patterns for probe slot selection results in quicker discovery through probe-probe overlaps.

This probabilistic approach essentially increases the possibility of discovery through a probe-probe overlap without diminishing a probe slot's ability to find an anchor slot in the symmetric case. However, in the asymmetric case, randomized probing increases the worst-case bound since in the worst case, the probe slot of the node with the bigger t can go through all regular slots first before meeting an anchor slot. In that case, the worst-case bound would be $t_A \cdot \lceil \frac{\lfloor \frac{t_A}{2} \rfloor}{2} \rceil$ slots, where t_A is the period of the node with bigger period length (smaller duty-cycle). However, the bound can be kept the same if we restrict the permutations that can be chosen. A simple approach would be to limit randomness within blocks of t_{base} slots but this would decrease average-case performance for symmetric duty cycles. We next present more intelligent permutations that would preserve the worst-case bound but allow more randomness.

3.8 Restricted Randomized Probing in the Asymmetric Case

The worst-case bound for sequential probing is based on the observation that A's probe slot is guaranteed to meet an anchor slot of B every t_B periods. However, no such guarantee can be provided for randomized probing. To better understand this point, let us divide the slots of A into t_B buckets. Assuming that the anchor slot is at position 0, the buckets would be as follows:
Bucket 1: $\{1, 1 + t_B, 1 + 2 \times t_B, ...\}$
Bucket 2: $\{2, 2 + t_B, 2 + 2 \times t_B, ...\}$
.....
Bucket t_B: $\{t_B, 2 \times t_B, 3 \times t_B, ...\}$

Based on the offset between A and B, B's anchor slots would fall into any of the above t_B buckets. For example, if the offset between A and B is 3 slots, then B's anchor slots with respect to A's anchor slot would be at slots 3, $3 + t_B$, $3+2\times t_B$ and so on (Bucket 3). With sequential probing, it is guaranteed that the movement pattern followed by the probe slot will contain one slot from each of the above buckets every t_B slots. Unfortunately, with randomized probing, no such guarantee can be provided. We next present a slightly restricted form of randomized probing that addresses this issue.

The basic goal of restricted randomized probing is to ensure that only those movement patterns would be chosen for probing where there is exactly one slot from each bucket every t_B slots. We achieve this goal with two-step randomization. First, a random permutation of the buckets gets chosen. This determines the order in which slots from different buckets would appear. The next step is to randomly choose a permutation of slots within each bucket. We illustrate the strategy with the following example.

Let us assume that $t_A = 60$ and $t_B = 10$. With sequential probing without striping, A's probe slot will follow the pattern (left to right, top to bottom):

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Each column represents one bucket. The first step of restricted randomized probing is to randomize the order of these buckets. One such valid ordering can be:

6	9	1	10	2	5	7	8	4	3
16	19	11	20	12	15	17	18	14	13
26	29	21	30	22	25	27	28	24	23

The second and final step is to randomize the order of slots within each bucket:

16	9	1	10	22	15	7	18	4	23
6	29	11	20	12	25	17	8	24	3
26	19	21	30	2	5	27	28	14	13

The above pattern ensures that there is exactly one slot

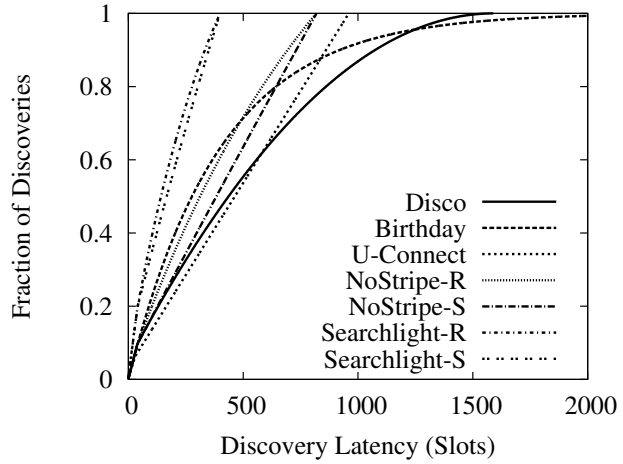

Figure 7: CDF for Discovery Latency - 5% duty cycle

from each bucket every t_B slots. Thus, by adopting this approach, restricted randomized probing achieves the same worst-case bound as sequential probing in the symmetric case.

3.9 Effects of clock drift

Searchlight's systematic probing relies on the assumption of a constant offset between the anchor slots of any two nodes. In reality, clocks drift and the offset between anchor slots will change over time. However, if the drift is sufficiently small, the worst case bound will not change.

To understand the effect of clock drift on the worst case bound, let us consider the worst case for two node's schedules. We will assume that no probe- probe discoveries occur and concentrate on probe-anchor discovery. In Section 3.2, we showed that this kind of discovery must exist given any two schedules. The least amount of overlap will occur between an anchor slot and a probe slot when the anchor slot starts immediately following the probe slot. Due to the striped nature of Searchlight, the anchor slot of size $x(1+\delta)$ will start at the same time as the inactive slot of size $x(1-\delta)$ producing a total overlap of $x \cdot (2\delta)$ with the next active slot.

Since this anchor-probe overlap is the only overlap that our worst case analysis relies upon, if the anchor slot were to drift to the point where it was no longer overlapping with this probe slot within one hyper-period, discovery could not be guaranteed. Thus the clock drift must be less than $\frac{x \cdot (2\delta)}{T}$ for the worst case bound to hold. Given this constraint, δ can be chosen based on the clock drift for a given device such that the bound holds.

4. EVALUATION

The primary goal of our evaluation is to show that Searchlight achieves significant performance improvements over other asynchronous neighbor discovery protocols by virtue of its systematic probing. Specifically, we evaluate how long it takes for different protocols to discover neighbors when *they spend the same amount of energy*, i.e., operate at the same duty cycle. The time from when two nodes first enter each other's transmission range to the time when they actually discover each other is known as the *discovery latency*. We look at the CDF of discovery latencies to understand the overall trend of a protocol. Since absolute latency in terms of time units is dependent on slot-width and slot-width is platform dependent and the same for all protocols, we use the number of slots as the metric for latency. We compare the performance of the variants of Searchlight with the deterministic protocols, Disco and U-Connect, and with the probabilistic Birthday protocol.

Since duty cycle directly translates to energy, we have not included separate results for energy consumption. Instead, we normalized all protocols to consume the same amount of energy and compared the performance of all of the protocols when they operated at the same duty cycle. Therefore, our results show how different protocols perform in terms of latency when they use the same amount of energy. The other possibility would be to hold the latency fixed and show how much energy is consumed to achieve that latency. However, this results are simply the inverse of the latency results.

For any random pair of nodes, the discovery latency can vary widely based on the relative offset. To fully characterize the performance of the different protocols, including the worst-case bound, we evaluated the protocols across all possible offsets through a state-based simulation.

Except for the Birthday protocol, all other protocols follow a discovery schedule to determine when to sleep and when to wake up. This schedule repeats every T slots, which we call the hyper-period. When a node is in a particular slot in its schedule, that slot index can be considered the *state* of the node at that point and is always in the range $[0, T-1]$. When two nodes with hyper periods T_1 and T_2 come into each other's transmission range, their states have one of $T_1 \cdot T_2$ possible combinations. For a given combination, the discovery latency is always the same. For Disco, U-Connect and Searchlight-S, we use this observation to loop through all possible combinations and determine the latency for each case. The same approach is not feasible for Searchlight-R since for a given t, a node can choose any of the $(\lfloor \frac{t}{2} \rfloor - 1)!$ permutations to determine the schedule of its probe slot. For scalability purposes, we run the protocol 1000 times with different seeds. At each run, we generate a new schedule for both nodes. Then, for that particular schedule pair, we loop through all possible state combinations like we do for Searchlight-S. Since for Disco and U-Connect, all slots are of equal size, we assume the slot boundaries to be completely aligned for the worst case (non-alignment leads to more overlap). On the other hand, since Searchlight uses "overflowing" active slots, complete slot boundary alignment creates the best case scenario. Therefore, we consider the slot boundaries to be offset by half a slot-width for simulating Searchlight. For the Birthday protocol, we use a closed form expression for determining the CDF and the expected value of discovery latency [21].

4.1 Symmetric case

First, we compare the performance of different protocols when nodes operate at the same duty cycle. We look at the cumulative distribution of discovery latencies for all protocols operating at a 5% duty cycle (see Figure 7). Based on the protocol design as described in respective papers to achieve a 5% duty cycle, Disco uses the primes (37,43) and U-Connect uses the prime 31. Birthday uses probability = 0.05. Both versions of Searchlight use the integer 40, which results in a 5% duty cycle as described earlier. To gauge the effectiveness of striped probing, we also evalu-

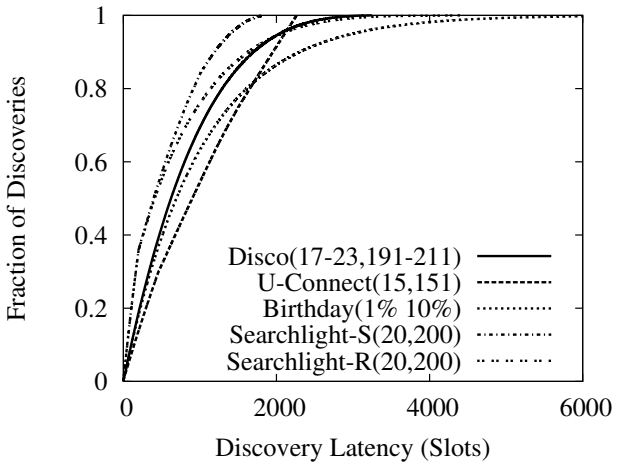

Figure 8: CDF of Discovery Latency - Asymmetric Duty Cycles - 1%-10%

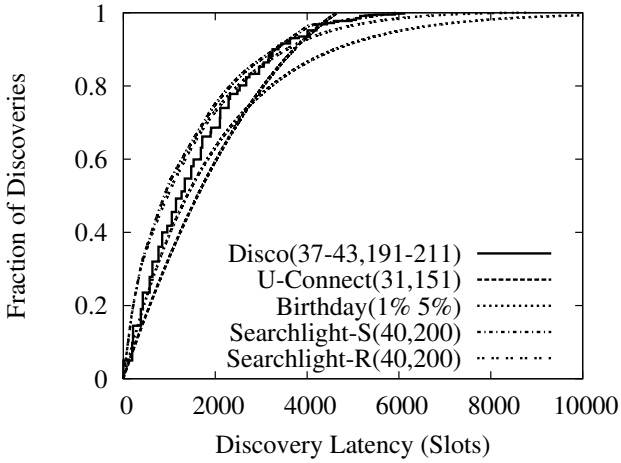

Figure 9: CDF of Discovery Latency - Asymmetric Duty Cycles - 1%-5%

ate both sequential and randomized versions of Searchlight without striped probing. They are denoted by NoStripe-S and NoStripe-R respectively. Both versions of Searchlight *always* achieve the lowest latency in the worst-case, with Searchlight-R doing better in the average case by virtue of its probabilistic component. Without striping, the number of slots needed for discovery doubles. Still, for 65% of the time, NoStripe-R performs on par with the Birthday protocol or slightly lags behind. Beyond that, the probabilistic nature of the Birthday protocol leads to a long tail and NoStripe-R achieves the lowest latency. In comparison to U-Connect, NoStripe-R achieves better latency all along. For both versions of Searchlight, maximum discovery latency is 400 slots, followed by 800 slots for the NoStripe versions. U-Connect comes closest with a maximum discovery latency of 961 slots, which is more than a 100% increase over Searchlight.

4.2 Asymmetric case

The evaluation of the asymmetric case is particularly important because the worst-case latency is different for Searchlight in this scenario. In this section, we evaluate two scenarios. In the first, one node operates at 1% duty cycle and the other at 10%. In the second, the second node operates at 5% duty cycle. As part of our future work, we plan to explore the space of asymmetry and how the degree of asymmetry affects all of the protocols.

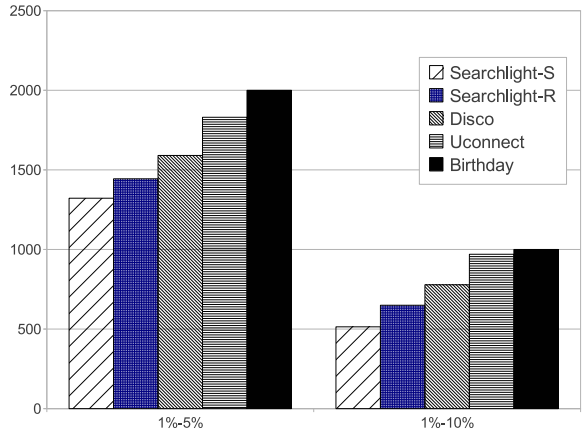

Figure 10: Average Latency - Asymmetric Duty Cycles - 1%-5% and 1%-10%

When two nodes operate at 1% and 10% duty cycles respectively (see Figure 8), Searchlight-S performs best all along and its worst-case discovery latency is almost half of U-Connect. Searchlight-R initially performs similar to Searchlight-S, but because of the higher worst-case bound starts to deviate after around the 65-th percentile.

For the asymmetric scenario when the two duty cycles are 1% and 5% (see Figure 9) U-Connect outperforms Searchlight-S in terms of worst-case discovery latency. In the asymmetric case, we know the worst-case bound for Searchlight-S is

$$\frac{(t_B + (\lceil \frac{\lfloor \frac{t_A}{2} \rfloor}{2} \rceil \mod t_B))}{2} \cdot t_A,$$

where t_A and t_B are the two periods and $t_A > t_B$. For the 1%-10% case, $\lfloor \frac{t_A}{2} \rfloor = 100$ was completely divisible by the smaller period $t_B = 20$, resulting in lowest worst-case latency. But for 1%-5%, 100 is not completely divisible by $t_B = 40$, which has led to the higher worst-case bound.

To further investigate this issue, we look at the average latency for both scenarios with 1%-5% and 1%-10% asymmetric duty cycles (see Figure 10). In both cases, Searchlight-S performs best, followed by Searchlight-R, while U-Connect performs even worse than Disco. This shows that in terms of average latency, Searchlight-S still performs best in all asymmetric scenarios.

4.2.1 *Effect of restricting Permutation*

We next evaluate the effect of restricting the set of allowable permutations for randomized probing. When two nodes operate at 1% and 10% duty cycle respectively, the worst-case latency for Searchlight-S is less than 50% of the worst-case latency for Searchlight-R (see Figure 8). But when we restrict the set of movement patterns that can be chosen, the worst-case performance of such restricted randomized probing (Searchlight-R+) becomes almost the same as Searchlight-S (see Figure 11).

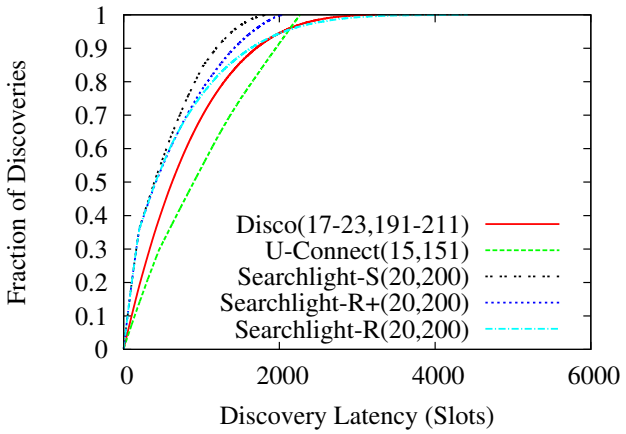

Figure 11: Effect of Restricted Randomized Probing: 1%-10% duty cycle

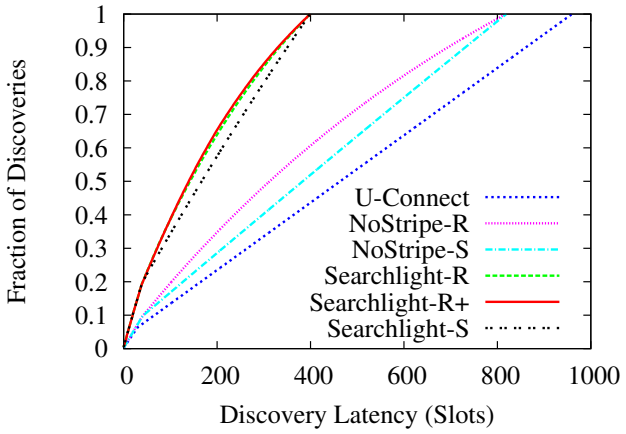

Figure 12: Effect of Restricted Randomized Probing: 5% duty cycle

While such restricted randomization helps in the asymmetric case, it is important to evaluate whether it decreases the advantage of Searchlight-R in the symmetric case. We next look at the comparative performance of Searchlight-R+ when all protocols operate at 5% duty cycle. Since 5% duty cycle corresponds to a period of 40 slots for Searchlight, we use 10 slots as the base period for restricting randomization. As we can see from Figure 12, Searchlight-R+ performs the same as Searchlight-R. This shows that the reduction in randomization does not affect the average-case behavior of randomized probing in the symmetric case.

5. IMPLEMENTATION

One of the main objectives for designing asynchronous neighbor discovery protocols is to facilitate proximity-based communication between handheld devices like smartphones. To gauge how Searchlight achieves this goal in practice, we implemented the protocol in Python and deployed it on the Nokia N900, a smartphone by Nokia that supports the open source Maemo 5 mobile device platform [27], and a Dell Latitude E6400 laptop with an Intel 5100AGN Wi-Fi card.

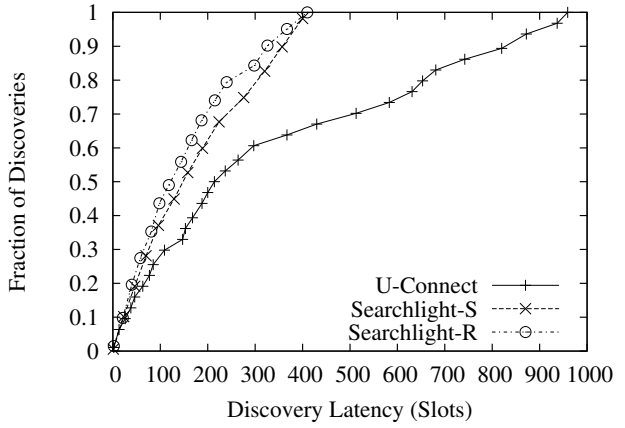

Figure 13: Implementation: CDF of Latency for Symmetric Duty Cycles

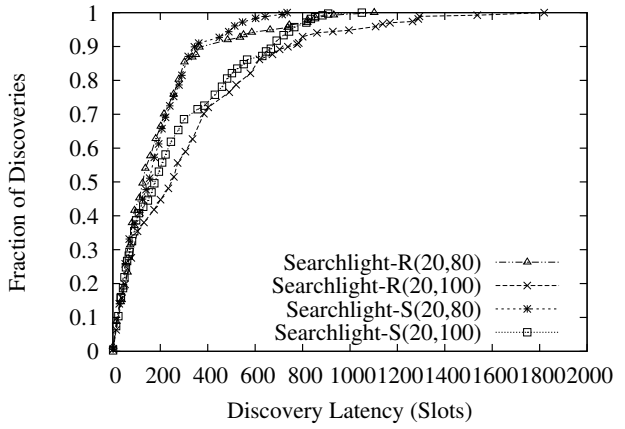

Figure 14: Implementation: CDF of Latency for Asymmetric

Results shown are collected strictly from the Nokia devices. We first describe different implementation issues and then present some preliminary results.

5.1 Implementation Issues

5.1.1 *Slot Duration*

We implemented Searchlight to use the Wi-Fi radio of the N900 phone for neighbor discovery. Earlier protocols for asynchronously discovering neighbors were all implemented on sensor nodes, allowing them to use small slots on the order of milliseconds [25] or even microseconds [11]. However, unlike sensor radios, Wi-Fi radios have a non-negligible transition latency from sleep to transmit/receive. On the N900 phone, we found out that from the application level, the time to bring the wireless interface up is around 1 second. Because of this latency, we chose a slot size of 2 seconds. Other phones we have looked at, including the Android G1, also have startup latencies on the order of seconds. These 2 second slots are quite large and so result in longer discovery delays. However, this is a limitation of the current hardware and not of our protocol. Although we expect this

delay to be improved in future generations of smartphones, our implementation evaluates the current state-of-the art.

5.1.2 Pre Slots

Because of the non-negligible start-up time, we introduced the notion of *pre* slots. A *pre* slot basically precedes any active slot and switches on the interface. For example, assume that a node needs to be active during slot 10. Now, if the command to wake up the radio is issued at the beginning of slot 10, it will take almost the whole slot duration for that command to return and the effective awake time during that slot will be reduced to 1 second. To get around this problem, the wake up command now gets issued at the start of the preceding slot which is slot 9 in this case. Such slots are called *pre* slots and their positions are determined based on the active slot schedule. *Pre* slots are kept shorter than normal slots to allow for the larger active slots. In U-Connect, two active slots can be neighbors. In this case, the radio is left on rather than using a *pre* slot.

5.1.3 Active slot

When the protocol starts up, it creates an active slot schedule based on the desired duty cycle. In an active slot, a hello message containing the node id gets sent at the very beginning and at the very end of the slot. In between, the node continuously listens for hello messages from other nodes. When it gets a hello message, it adds the name of the sender to a discovered list. The size of the slot extension for active slots, which guarantees slot overlap, was set to be 5 milliseconds. This more than accommodates a hello message transmission and jitter from timer scheduling. As newer smartphones improve device wakeup times, we will evaluate the use of shorter slots and also less overflow, with the goal of maintaining enough overlap to compensate for completely aligned slots with requiring large slot sizes.

5.2 Evaluation

We implemented Searchlight-S and Searchlight-R on five N900 phones and logged the discovery latency for around 200 runs with 5% duty cycle. We also implemented U-Connect since it performs best among existing protocols in the symmetric case. Since U-Connect has consecutive active slots, we implemented a striped version of U-Connect, which reduces the number of active slots by about 10%.

For the implementations, Searchlight-R and Searchlight-S fare much better than U-Connect (see Figure 13), similar to our simulation results (see Figure 7). Even with striped probing, U-Connect's still falls short of Searchlight. We also looked at the asymmetric case of Searchlight with t values of (20,80) and (20,100) which represent 10%, 2.5%, 10%, and 2% duty cycles respectively. Searchlight-S outperforms its randomized version as was predicted in the asymmetric case (see Figure 14).

The implementation results actually turned out to be better than the state-based simulation results, with more discoveries taking place at lower latencies. Since the devices were not synchronized, this is probably an artifact of the slots being non-aligned (see section 3.3). However, the overall trends agree with the simulation results.

6. CONCLUSIONS AND FUTURE WORK

Solving the problem of energy efficient asynchronous neighbor discovery is an important pre-condition for more widespread use of proximity-based communication between mobile devices, including sensors and smartphones. In this paper, we present Searchlight, a new asynchronous neighbor discovery protocol. By adopting a systematic approach that has both deterministic and probabilistic components, Searchlight achieves average-case performance comparable to the probabilistic protocols and significantly improves worst-case bounds for symmetric operation in comparison to the current best deterministic protocols. In the asymmetric case, Searchlight still has the best performance. However, if multiple devices share the same duty cycle, even when other devices use different duty cycles, Searchlight exploits this partial symmetry and provides even better performance. Searchlight was also successfully implemented on a smartphone testbed, and showed performance trends similar to the simulation results.

As future work, we would like to integrate neighbor discovery with data transmission. It is not clear when a node should send a packet to an already discovered neighbor. Should it wait for the next overlap to take place and send a packet after receiving a beacon from the node it wishes to communicate with? This will obviously increase delay and we will have to consider the energy-latency trade-off.

Finally, we would like to investigate how Searchlight can dynamically adapt to energy requirements, contact patterns and other factors to adjust its duty cycle. For example, when neighbor count is low, a higher duty cycle might be required to find enough neighbors that meet application requirements. On the other hand, in a crowded place where the number of co-located nodes is high, a lower duty cycle might suffice to find a reasonable number of neighbors.

7. REFERENCES

[1] "Google latitude," http://www.google.com/latitude.
[2] "Gowalla," http://gowalla.com.
[3] "Foursquare," http://foursquare.com.
[4] "Facebook places," http://www.facebook.com/about/location.
[5] "Who's near me," http://www.synergetechsolutions.com/whos-near-me.aspx.
[6] A. K. Pietiläinen, E. Oliver, J. Lebrun, G. Varghese, and C. Diot, "MobiClique: middleware for mobile social networking," in *WOSN '09: Proceedings of the 2nd ACM workshop on Online social networks*, 2009, pp. 49–54.
[7] "Aka-aki," http://akakicom.
[8] "Bluehoo," http://bluehoo.com.
[9] "Nintendo 3ds - streetpass." http://www.nintendo.com/3ds/hardware.
[10] "Sony ps vita - near." http://us.playstation.com/psvita.
[11] A. Kandhalu, K. Lakshmanan, and R. R. Rajkumar, "U-connect: a low-latency energy-efficient asynchronous neighbor discovery protocol," in *Proceedings of IPSN '10*, 2010, pp. 350–361.
[12] "Qualcomm alljoyn," https://www.alljoyn.org/.
[13] "Android - an open handset alliance project." http://developer.android.com.
[14] "Wi-fi direct," http://www.wi-fi.org/discover-and-learn/wi-fi-direct

[15] T. Liu, C. M. Sadler, P. Zhang, and M. Martonosi, "Implementing software on resource-constrained mobile sensors: experiences with impala and zebranet," in *MobiSys 2004*. ACM, 2004, pp. 256–269.

[16] H. Jun, M. H. Ammar, M. D. Corner, and E. W. Zegura, "Hierarchical power management in disruption tolerant networks using traffic-aware optimization," *Comput. Commun.*, vol. 32, pp. 1710–1723, October 2009.

[17] J. Elson and K. Römer, "Wireless sensor networks: a new regime for time synchronization," *SIGCOMM Comput. Commun. Rev.*, vol. 33, pp. 149–154, January 2003.

[18] J. Paek, J. Kim, and R. Govindan, "Energy-efficient rate-adaptive gps-based positioning for smartphones," in *Proceedings of MobiSys '10*. ACM, 2010, pp. 299–314.

[19] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *Proceedings of SenSys '04*. ACM, 2004, pp. 95–107.

[20] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient mac protocol for wireless sensor networks," in *INFOCOM 2002*, vol. 3, 2002, pp. 1567–1576 vol.3.

[21] M. J. McGlynn and S. A. Borbash, "Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks," in *Proceedings of MobiHoc '01*. ACM, 2001, pp. 137–145.

[22] S. Lai, B. Ravindran, and H. Cho, "Heterogenous quorum-based wakeup scheduling in wireless sensor networks," *IEEE Transactions on Computers*, vol. 99, no. PrePrints, 2010.

[23] Y.-C. Tseng, C.-S. Hsu, and T.-Y. Hsieh, "Power-saving protocols for ieee 802.11-based multi-hop ad hoc networks," in *INFOCOM*, 2002.

[24] R. Zheng, J. C. Hou, and L. Sha, "Asynchronous wakeup for ad hoc networks," in *MobiHoc 2003*. ACM, 2003, pp. 35–45.

[25] P. Dutta and D. Culler, "Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications," in *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, 2008, pp. 71–84.

[26] I. Niven and H. S. Zuckerman, *An Introduction to the Theory of Numbers*. John Wiley and Sons (WIE), 1991.

[27] "Maemo - open source operating system." http://maemo.org.